Chapter 4 has a lot of definitions and formulas, and it's easy to lose track of how they're all related to each other. This handout is an attempt to show you the big picture; it also helps you figure out what I think is most important about the chapter before our exam next week. After a summary of noisy coding, I added a section about the notation and language in this chapter, which has been confusing for some people.

The Big Picture

Whenever we transmit information using cables, radio waves, or any other method, errors occur. There are plenty of everyday examples: static on a telephone line or a cell phone, a fuzzy television picture, or bad reception on a radio. The basic question is this: given a transmission channel with a known error rate, is it possible to encode our information in such a way that we can recover from the errors which will inevitably occur?

For the rest of this summary, we'll assume all of our codes are binary. In an effort to avoid confusion, "bit" will always mean the unit of information (see below), and I'll refer to the 0's and 1's in codewords as "digits."

Error Detection and Correction. By "recover from the errors" above I really mean, "Can we detect or even correct the errors?" We detect errors by adding redundancy – parity digits, CRCs, etc. – which let us know when certain errors have occurred. Then we can toss out the bad word and ask for it to be resent. If we add even more redundancy we can automatically correct errors as we move along.

Why wouldn't we always just implement an error-*correcting* scheme and avoid the hassle of having words retransmitted? The truth is that error correction can require a *lot* more redundancy than error detection; to reliably detect errors might require adding a few digits (0's or 1's) per codeword, whereas a good error correction scheme might nearly double the size of your codewords. If you're fairly confident that the channel has a low error rate, it might be more efficient in the long run to use shorter words which offer some error-detection, and just accept the fact that some words will be resent.

It's worth mentioning that no error detection or correction scheme is perfect. If so many errors occur during transmission that a codeword is transformed into another valid codeword, this will never be detected. Similarly, if enough errors occur a received word could be "corrected" to the wrong word. The point is to know enough to minimize this possibility.

Information. Recall that our basic unit of information is a "bit." We first defined it in terms of heads or tails with a fair coin, but it's just as easy to use 0 and 1 instead of H and T. In class I've often used "Yes" and "No" instead of 0 and 1. So you can think of it this way: a bit is exactly enough information to answer a Yes/No question where each answer is equally likely.

Rate of a Code. A binary code has codewords which are strings of 0's and 1's. We defined the rate of a code as[1]

$$\text{Rate of } f = \frac{\log_2(\#\text{ of codewords})}{\text{length of longest codeword}}$$

[1]Remember, the codewords often all have the same length, but this isn't always true, as we saw with Huffman codes.

The rate gives us a way to tell how efficiently information is encoded by f. The unit here is "bits per digit in the codewords." Let's think about two examples from class to make this clearer.

(1) Suppose we want to send an answer to a Yes/No question, but we're worried that the 0 or 1 might be corrupted by the transmission channel. So we create a repetition code with two words, 00000 and 11111. (If something else is received, the decoder will will a "majority vote" algorithm.) You can double check that the rate of this code is 0.2. That makes perfect sense, because we're using five digits to encode one bit of information, so each digit in a codeword has 1/5 of a bit of information.

Equivalently, if each digit in the codeword has 0.2 bits of information, 5 digits gives $5 \cdot 0.2 = 1$ bit of information, i.e. the answer "Yes" or "No."

(2) Now suppose I want to sent the answer to a multiple choice question with 4 answers. I create a code with the four words $\{00, 01, 10, 11\}$. You can check that the rate of this code is 1. This means each digit in a codeword represents a full bit of information. Equivalently, each two-digit word encodes two bits of information. That's exactly the amount of information I need to specify the four different answers.[2]

Note that this code has *no* redundancy whatsoever! I need at least two digits to specify four possibilities, and I'm not using any more than two digits. Any errors would be undetectable; if 10 is received as 01, that's still a valid codeword! So codes with rate 1 are a poor choice unless I'm using a transmission channel with no errors at all. (That's *Noiseless* Coding, in the previous chapter.)

Channel Capacity and the Noisy Coding Theorem. Now that we know what the rate of a code is, we have to find a way to relate it to a given transmission channel. Roughly speaking, the *capacity* of a channel is the amount of information–measured in bits per digit–which can reliably be transmitted across a channel. **This is not the definition of capacity, but rather an *interpretation* of capacity made possibly by the Noisy Coding Theorem**. The actual definition was using intuitive notions of entropy and information, although in the important case of a Binary Symmetric Channel with error probability p, we have the handy formula:

$$\text{capacity } = 1 + p \log_2 p + (1-p) \log_2 (1-p)$$

The basic rule of noisy coding is that you should always use a code whose rate is less than the capacity of the transmission channel. If you have a noisy channel with capacity 0.23, you can only hope to reliably transmit 0.23 bits of information per digit. If you try to use a code with a rate of, say, 0.4 bits per digit, then your decoded text will inevitably be full of errors. On the other hand, our repetition code from above, with a rate of 0.2 bits per digit, could be appropriate here. Errors will occur during transmission, but the decoder should still be able to recover the correct word. ("Should" could be made more precise using probability....)

For a given Binary Symmetric Channel (with $p < \frac{1}{2}$), the *Noisy Coding Theorem* tells us that we can find codes with rates which (1) are arbitrarily close to the capacity and (2) have a probability of error in decoding which is virtually zero. (In other words, it's almost certain that the message which is sent is received as intended, and it's done with as efficient of a rate as we could hope for given that channel.) These statements are all made more precise in the book; I'm only worried about the big ideas here. Unfortunately, the Theorem doesn't tell us how to *make* these codes, so that's what everybody in the field is trying to figure out.

[2]Each bit represents two possibilities, so two bits result in $2^2 = 4$ possibilities.

It's also worth mentioning the converse of the Noisy Coding Theorem, which doesn't appear in your textbook. This says that if the rate of the code is higher than the capacity, you can't hope to reliably transmit your information across the channel; the probability of errors will be too high and your message will be decoded to gobbledygook. (This is what allows for the "interpretation" of capacity mentioned above.)

Slippery Notation

One of the things that makes this chapter tricky is that the meaning of certain notations change. We've talked in class about the phrase "Abuse of Language." This means we're saying (or writing) something which doesn't quite fit our definitions, but we think it's pretty clear what it means. Here's a standard example using one of the simplest functions from calculus:

$$f : \mathbb{R} \to \mathbb{R}$$
$$x \mapsto x^2$$

Then the *function* is f, and the notation $f(x)$ really means "the image of x under the function f." Yet we always talk about "the function $f(x)$." Technically this is incorrect, and yet it makes a lot of sense to us.

In this chapter there are lots of things which don't quite fit with earlier definitions, but "hopefully" make sense. (Well, everything makes sense once you understand it....) Here are a few things that come to mind. Hopefully in the end you'll agree that it's easier to use these abuses of language than to redefine all of the terms.

- Earlier we defined a random variable X to be a *real*-valued function from a sample space, i.e. $X : \Omega \to \mathbb{R}$. In the coding chapters the random variables often output letters or words instead of real numbers. We could get around this by letting (for example) X take on values $1, 2, \ldots, 26$ instead of $a, b, \ldots, z$, but it seems easier to think of the letters instead.
- A memoryless source X is a sequence of randon variables $X_1, X_2, \ldots$, each of which emits the same words with given probabilities: $P(X_i \text{ emits } w_1) = p_1$, and so on. Intuitively, X just keeps emitting words from the same set of source words, so we often drop the subscripts and talk about $P(X \text{ emits } w_1)$, even if this isn't quite correct.
- A channel has an input character and an output character, but usually we're interested in sending more than just a single character. (For example, if the inputs and outputs are all 0's and 1's, we might be sending dozens, if not hundreds or thousands, of characters across our channel.) Technically that means we're talking about an *extension* of the channel (see page 69), but we hardly ever explicitly mention the extension.
- Sources sometimes show up in different contexts. In the picture on page 62, X is a source emitting codewords. These words are then encoded into 0's and 1's and sent across a channel. In most of the rest of the chapter, however, those two steps are combined, and X is a source emitting 0's and 1's. Whatever coding (Huffman? Repetition?) is used was done before X came along. Y, on the other hand, generally refers to the words received at the other end of the transmission channel. These may or may not be valid codewords; if not, we'll either attempt to detect errors and ask for a retransmission, or try to automatically correct the received words so that they are valid codewords.